Other algorithms have been proposed that use ideas similar to those used in knapsack cryptosystems, but these too have been broken. The Lu-Lee cryptosystem [990,13] was broken in [20,614,873]; a modification [507] is also insecure [1620]. Attacks on the Goodman-McAuley cryptosystem are in [646,647,267,268]. The Pieprzyk cryptosystem [1246] can be broken by similar attacks. The Niemi cryptosystem [1169], based on modular knapsacks, was broken in [345,788]. A newer multistage knapsack [747] has not yet been broken, but I am not optimistic. Another variant is [294].

While a variation of the knapsack algorithm is currently secure—the Chor-Rivest knapsack [356], despite a "specialized attack" [743]—the amount of computation required makes it far less useful than the other algorithms discussed here. A variant, called the Powerline System, is not secure [958]. Most important, considering the ease with which all the other variations fell, it doesn't seem prudent to trust them.

### *Patents*

The original Merkle-Hellman algorithm is patented in the United States [720] and worldwide (see Table 19.1). Public Key Partners (PKP) licenses the patent, along with other public-key cryptography patents (see Section 25.5). The U.S. patent will expire on August 19, 1997.

## 19.3 RSA

Soon after Merkle's knapsack algorithm came the first full-fledged public-key algorithm, one that works for encryption and digital signatures: RSA [1328,1329]. Of all the public-key algorithms proposed over the years, RSA is by far the easiest to understand and implement. (Martin Gardner published an early description of the algorithm in his "Mathematical Games" column in *Scientific American* [599].) It is

**Table 19.1**
**Foreign Merkle-Hellman Knapsack Patents**

| Country | Number | Date of Issue |
|---|---|---|
| Belgium | 871039 | 5 Apr 1979 |
| Netherlands | 7810063 | 10 Apr 1979 |
| Great Britain | 2006580 | 2 May 1979 |
| Germany | 2843583 | 10 May 1979 |
| Sweden | 7810478 | 14 May 1979 |
| France | 2405532 | 8 Jun 1979 |
| Germany | 2843583 | 3 Jun 1982 |
| Germany | 2857905 | 15 Jul 1982 |
| Canada | 1128159 | 20 Jul 1982 |
| Great Britain | 2006580 | 18 Aug 1982 |
| Switzerland | 63416114 | 14 Jan 1983 |
| Italy | 1099780 | 28 Sep 1985 |

also the most popular. Named after the three inventors—Ron Rivest, Adi Shamir, and Leonard Adleman—it has since withstood years of extensive cryptanalysis. Although the cryptanalysis neither proved nor disproved RSA's security, it does suggest a confidence level in the algorithm.

RSA gets its security from the difficulty of factoring large numbers. The public and private keys are functions of a pair of large (100 to 200 digits or even larger) prime numbers. Recovering the plaintext from the public key and the ciphertext is conjectured to be equivalent to factoring the product of the two primes.

To generate the two keys, choose two random large prime numbers, $p$ and $q$. For maximum security, choose $p$ and $q$ of equal length. Compute the product:

$$n = pq$$

Then randomly choose the encryption key, $e$, such that $e$ and $(p-1)(q-1)$ are relatively prime. Finally, use the extended Euclidean algorithm to compute the decryption key, $d$, such that

$$ed \equiv 1 \bmod (p-1)(q-1)$$

In other words,

$$d = e^{-1} \bmod ((p-1)(q-1))$$

Note that $d$ and $n$ are also relatively prime. The numbers $e$ and $n$ are the public key; the number $d$ is the private key. The two primes, $p$ and $q$, are no longer needed. They should be discarded, but never revealed.

To encrypt a message $m$, first divide it into numerical blocks smaller than $n$ (with binary data, choose the largest power of 2 less than $n$). That is, if both $p$ and $q$ are 100-digit primes, then $n$ will have just under 200 digits and each message block, $m_i$, should be just under 200 digits long. (If you need to encrypt a fixed number of blocks, you can pad them with a few zeros on the left to ensure that they will always be less than $n$.) The encrypted message, $c$, will be made up of similarly sized message blocks, $c_i$, of about the same length. The encryption formula is simply

$$c_i = m_i^e \bmod n$$

To decrypt a message, take each encrypted block $c_i$ and compute

$$m_i = c_i^d \bmod n$$

Since

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i * 1 = m_i; \text{ all (mod } n)$$

the formula recovers the message. This is summarized in Table 19.2.

The message could just as easily have been encrypted with $d$ and decrypted with $e$; the choice is arbitrary. I will spare you the number theory that proves why this works; most current texts on cryptography cover it in detail.

A short example will probably go a long way to making this clearer. If $p = 47$ and $q = 71$, then

**Table 19.2**
**RSA Encryption**

*Public Key:*
$n$ product of two primes, $p$ and $q$ ($p$ and $q$ must remain secret)
$e$ relatively prime to $(p-1)(q-1)$

*Private Key:*
$d$ $e^{-1} \bmod ((p-1)(q-1))$

*Encrypting:*
$c = m^e \bmod n$

*Decrypting:*
$m = c^d \bmod n$

$$n = pq = 3337$$

The encryption key, $e$, must have no factors in common with

$$(p-1)(q-1) = 46 \cdot 70 = 3220$$

Choose $e$ (at random) to be 79. In that case

$$d = 79^{-1} \bmod 3220 = 1019$$

This number was calculated using the extended Euclidean algorithm (see Section 11.3). Publish $e$ and $n$, and keep $d$ secret. Discard $p$ and $q$.

To encrypt the message

$$m = 6882326879666683$$

first break it into small blocks. Three-digit blocks work nicely in this case. The message is split into six blocks, $m_i$, in which

$$m_1 = 688$$
$$m_2 = 232$$
$$m_3 = 687$$
$$m_4 = 966$$
$$m_5 = 668$$
$$m_6 = 003$$

The first block is encrypted as

$$688^{79} \bmod 3337 = 1570 = c_1$$

Performing the same operation on the subsequent blocks generates an encrypted message:

$$c = 1570\ 2756\ 2091\ 2276\ 2423\ 158$$

Decrypting the message requires performing the same exponentiation using the decryption key of 1019, so

$$1570^{1019} \bmod 3337 = 688 = m_1$$

The rest of the message can be recovered in this manner.

### RSA in Hardware

Much has been written on the subject of hardware implementations of RSA [1314, 1474,1456,1316,1485,874,1222,87,1410,1409,1343,998,367,1429,523,772]. Good survey articles are [258,872]. Many different chips perform RSA encryption [1310,252, 1101,1317,874,69,737,594,1275,1563,509,1223]. A partial list of currently available RSA chips, from [150,258], is listed in Table 19.3. Not all are available on the open market.

### Speed of RSA

In hardware, RSA is about 1000 times slower than DES. The fastest VLSI hardware implementation for RSA with a 512-bit modulus has a throughput of 64 kilobits per second [258]. There are also chips that perform 1024-bit RSA encryption. Currently chips are being planned that will approach 1 megabit per second using a 512-bit modulus; they will probably be available in 1995. Manufacturers have also implemented RSA in smart cards; these implementations are slower.

In software, DES is about 100 times faster than RSA. These numbers may change slightly as technology changes, but RSA will never approach the speed of symmetric algorithms. Table 19.4 gives sample software speeds of RSA [918].

### Software Speedups

RSA encryption goes much faster if you're smart about choosing a value of $e$. The three most common choices are 3, 17, and 65537 ($2^{16}+1$). (The binary representation of 65537 has only two ones, so it takes only 17 multiplications to exponentiate.) X.509 recommends 65537 [304], PEM recommends 3 [76], and PKCS #1 (see Section 24.14) recommends 3 or 65537 [1345]. There are no security problems with using

**Table 19.3**
**Existing RSA Chips**

| Company | Clock Speed | Baud Rate Per 512 Bits | Clock Cycles Per 512 Bit Encryption | Technology | Bits per Chip | Number of Transistors |
|---|---|---|---|---|---|---|
| Alpha Techn. | 25 MHz | 13 K | .98 M | 2 micron | 1024 | 180,000 |
| AT&T | 15 MHz | 19 K | .4 M | 1.5 micron | 298 | 100,000 |
| British Telecom | 10 MHz | 5.1 K | 1 M | 2.5 micron | 256 | — |
| Business Sim. Ltd. | 5 MHz | 3.8 K | .67 M | Gate Array | 32 | — |
| Calmos Syst. Inc. | 20 MHz | 28 K | .36 M | 2 micron | 593 | 95,000 |
| CNET | 25 MHz | 5.3 K | 2.3 M | 1 micron | 1024 | 100,000 |
| Cryptech | 14 MHz | 17 K | .4 M | Gate Array | 120 | 33,000 |
| Cylink | 30 MHz | 6.8 K | 1.2 M | 1.5 micron | 1024 | 150,000 |
| GEC Marconi | 25 MHz | 10.2 K | .67 M | 1.4 micron | 512 | 160,000 |
| Pijnenburg | 25 MHz | 50 K | .256 M | 1 micron | 1024 | 400,000 |
| Sandia | 8 MHz | 10 K | .4 M | 2 micron | 272 | 86,000 |
| Siemens | 5 MHz | 8.5 K | .3 M | 1 micron | 512 | 60,000 |

**Table 19.4**
**RSA Speeds for Different Modulus Lengths with an 8-bit Public Key (on a SPARC II)**

| | 512 bits | 768 bits | 1,024 bits |
|---|---|---|---|
| Encrypt | 0.03 sec | 0.05 sec | 0.08 sec |
| Decrypt | 0.16 sec | 0.48 sec | 0.93 sec |
| Sign | 0.16 sec | 0.52 sec | 0.97 sec |
| Verify | 0.02 sec | 0.07 sec | 0.08 sec |

any of these three values for $e$ (assuming you pad messages with random values—see later section), even if a whole group of users uses the same value for $e$.

Private key operations can be speeded up with the Chinese remainder theorem if you save the values of $p$ and $q$, and additional values such as $d \bmod (p-1)$, $d \bmod (q-1)$, and $q^{-1} \bmod p$ [1283,1276]. These additional numbers can easily be calculated from the private and public keys.

### *Security of RSA*

The security of RSA depends wholly on the problem of factoring large numbers. Technically, that's a lie. It is *conjectured* that the security of RSA depends on the problem of factoring large numbers. It has never been mathematically proven that you need to factor $n$ to calculate $m$ from $c$ and $e$. It is conceivable that an entirely different way to cryptanalyze RSA might be discovered. However, if this new way allows the cryptanalyst to deduce $d$, it could also be used as a new way to factor large numbers. I wouldn't worry about it too much.

It is also possible to attack RSA by guessing the value of $(p-1)(q-1)$. This attack is no easier than factoring $n$ [1616].

For the ultraskeptical, some RSA variants have been proved to be as difficult as factoring (see Section 19.5). Also look at [36], which shows that recovering even certain bits of information from an RSA-encrypted ciphertext is as hard as decrypting the entire message.

Factoring $n$ is the most obvious means of attack. Any adversary will have the public key, $e$, and the modulus, $n$. To find the decryption key, $d$, he has to factor $n$. Section 11.4 discusses the current state of factoring technology. Currently, a 129-decimal-digit modulus is at the edge of factoring technology. So, $n$ must be larger than that. Read Section 7.2 on public key length.

It is certainly possible for a cryptanalyst to try every possible $d$ until he stumbles on the correct one. This brute-force attack is even less efficient than trying to factor $n$.

From time to time, people claim to have found easy ways to break RSA, but to date no such claim has held up. For example, in 1993 a draft paper by William Payne proposed a method based on Fermat's little theorem [1234]. Unfortunately, this method is also slower than factoring the modulus.

There's another worry. Most common algorithms for computing primes $p$ and $q$ are probabilistic; what happens if $p$ or $q$ is composite? Well, first you can make the odds of that happening as small as you want. And if it does happen, the odds are that

encryption and decryption won't work properly—you'll notice right away. There are a few numbers, called Carmichael numbers, which certain probabilistic primality algorithms will fail to detect. These are exceedingly rare, but they are insecure [746]. Honestly, I wouldn't worry about it.

### Chosen Ciphertext Attack against RSA

Some attacks work against the implementation of RSA. These are not attacks against the basic algorithm, but against the protocol. It's important to realize that it's not enough to use RSA. Details matter.

*Scenario 1:* Eve, listening in on Alice's communications, manages to collect a ciphertext message, $c$, encrypted with RSA in her public key. Eve wants to be able to read the message. Mathematically, she wants $m$, in which

$$m = c^d$$

To recover $m$, she first chooses a random number, $r$, such that $r$ is less than $n$. She gets Alice's public key, $e$. Then she computes

$$x = r^e \bmod n$$
$$y = xc \bmod n$$
$$t = r^{-1} \bmod n$$

If $x = r^e \bmod n$, then $r = x^d \bmod n$.

Now, Eve gets Alice to sign $y$ with her private key, thereby decrypting $y$. (Alice has to sign the message, not the hash of the message.) Remember, Alice has never seen $y$ before. Alice sends Eve

$$u = y^d \bmod n$$

Now, Eve computes

$$tu \bmod n = r^{-1}y^d \bmod n = r^{-1}x^d c^d \bmod n = c^d \bmod n = m$$

Eve now has $m$.

*Scenario 2:* Trent is a computer notary public. If Alice wants a document notarized, she sends it to Trent. Trent signs it with an RSA digital signature and sends it back. (No one-way hash functions are used here; Trent encrypts the entire message with his private key.)

Mallory wants Trent to sign a message he otherwise wouldn't. Maybe it has a phony timestamp; maybe it purports to be from another person. Whatever the reason, Trent would never sign it if he had a choice. Let's call this message $m'$.

First, Mallory chooses an arbitrary value $x$ and computes $y = x^e \bmod n$. He can easily get $e$; it's Trent's public key and must be public to verify his signatures. Then he computes $m = ym' \bmod n$, and sends $m$ to Trent to sign. Trent returns $m'^d \bmod n$. Now Mallory calculates $(m^d \bmod n)x^{-1} \bmod n$, which equals $n'^d \bmod n$ and is the signature of $m'$.

Actually, Mallory can use several methods to accomplish these same things [423,458,486]. The weakness they all exploit is that exponentiation preserves the multiplicative structure of the input. That is:

$$(xm)^d \bmod n = x^d m^d \bmod n$$

*Scenario 3:* Eve wants Alice to sign $m_3$. She generates two messages, $m_1$ and $m_2$, such that

$$m_3 \equiv m_1 m_2 \pmod{n}$$

If Eve can get Alice to sign $m_1$ and $m_2$, she can calculate $m_3$:

$$m_3{}^d = (m_1{}^d \bmod n)(m_2{}^d \bmod n)$$

Moral: Never use RSA to sign a random document presented to you by a stranger. Always use a one-way hash function first. The ISO 9796 block format prevents this attack.

### *Common Modulus Attack on RSA*

A possible RSA implementation gives everyone the same $n$, but different values for the exponents $e$ and $d$. Unfortunately, this doesn't work. The most obvious problem is that if the same message is ever encrypted with two different exponents (both having the same modulus), and those two exponents are relatively prime (which they generally would be), then the plaintext can be recovered without either of the decryption exponents [1457].

Let $m$ be the plaintext message. The two encryption keys are $e_1$ and $e_2$. The common modulus is $n$. The two ciphertext messages are:

$$c_1 = m^{e_1} \bmod n$$
$$c_2 = m^{e_2} \bmod n$$

The cryptanalyst knows $n$, $e_1$, $e_2$, $c_1$, and $c_2$. Here's how he recovers $m$.

Since $e_1$ and $e_2$ are relatively prime, the extended Euclidean algorithm can find $r$ and $s$, such that

$$re_1 + se_2 = 1$$

Assuming $r$ is negative (either $r$ or $s$ has to be, so just call the negative one $r$), then the extended Euclidean algorithm can be used again to calculate $c_1{}^{-1}$. Then

$$(c_1{}^{-1})^{-r} \cdot c_2{}^s = m \bmod n$$

There are two other, more subtle, attacks against this type of system. One attack uses a probabilistic method for factoring $n$. The other uses a deterministic algorithm for calculating someone's secret key without factoring the modulus. Both attacks are described in detail in [449].

Moral: Don't share a common $n$ among a group of users.

### *Low Encryption Exponent Attack against RSA*

RSA encryption and signature verification are faster if you use a low value for $e$, but that can also be insecure [704]. If you encrypt $e(e+1)/2$ linearly dependent messages with different public keys having the same value of $e$, there is an attack against the system. If there are fewer than that many messages, or if the messages are unrelated, there is no problem. If the messages are identical, then $e$ messages are enough. The easiest solution is to pad messages with independent random values.

This also ensures that $m^e \bmod n \neq m^e$. Most real-world RSA implementations—PEM and PGP (see Sections 24.10 and 24.12), for example—do this.

Moral: Pad messages with random values before encrypting them; make sure $m$ is about the same size as $n$.

### *Low Decryption Exponent Attack against RSA*

Another attack, this one by Michael Wiener, will recover $d$, when $d$ is up to one quarter the size of $n$ and $e$ is less than $n$ [1596]. This rarely occurs if $e$ and $d$ are chosen at random, and cannot occur if $e$ has a small value.

Moral: Choose a large value for $d$.

### *Lessons Learned*

Judith Moore lists several restrictions on the use of RSA, based on the success of these attacks [1114,1115]:

- Knowledge of one encryption/decryption pair of exponents for a given modulus enables an attacker to factor the modulus.
- Knowledge of one encryption/decryption pair of exponents for a given modulus enables an attacker to calculate other encryption/decryption pairs without having to factor $n$.
- A common modulus should not be used in a protocol using RSA in a communications network. (This should be obvious from the previous two points.)
- Messages should be padded with random values to prevent attacks on low encryption exponents.
- The decryption exponent should be large.

Remember, it is not enough to have a secure cryptographic algorithm. The entire cryptosystem must be secure, and the cryptographic protocol must be secure. A failure in any of those three areas makes the overall system insecure.

### *Attack on Encrypting and Signing with RSA*

It makes sense to sign a message before encrypting it (see Section 2.7), but not everyone follows this practice. With RSA, there is an attack against protocols that encrypt before signing [48].

Alice wants to send a message to Bob. First she encrypts it with Bob's public key; then she signs it with her private key. Her encrypted and signed message looks like:

$$(m^{e_B} \bmod n_B)^{d_A} \bmod n_A$$

Here's how Bob can claim that Alice sent him $m'$ and not $m$. Realize that since Bob knows the factorization of $n_B$ (it's his modulus), he can calculate discrete logarithms with respect to $n_B$. Therefore, all he has to do is to find an $x$ such that

$$m'^x = m \bmod n_B$$

Then, if he can publish $xe_B$ as his new public exponent and keep $n_B$ as his modulus, he can claim that Alice sent him message $m'$ encrypted in this new exponent.

This is a particularly nasty attack in some circumstances. Note that hash functions don't solve the problem. However, forcing a fixed encryption exponent for every user does.

***Standards***

RSA is a *de facto* standard in much of the world. The ISO almost, but not quite, created an RSA digital-signature standard; RSA is in an information annex to ISO 9796 [762]. The French banking community standardized on RSA [525], as have the Australians [1498]. The United States currently has no standard for public-key encryption, because of pressure from the NSA and patent issues. Many U.S. companies use PKCS (see Section 24.14), written by RSA Data Security, Inc. A draft ANSI banking standard specifies RSA [61].

***Patents***

The RSA algorithm is patented in the United States [1330], but not in any other country. PKP licenses the patent, along with other public-key cryptography patents (see Section 25.5). The U.S. patent will expire on September 20, 2000.

## 19.4 POHLIG-HELLMAN

The Pohlig-Hellman encryption scheme [1253] is similar to RSA. It is not a symmetric algorithm, because different keys are used for encryption and decryption. It is not a public-key scheme, because the keys are easily derivable from each other; both the encryption and decryption keys must be kept secret.

Like RSA,

$$C = P^e \bmod n$$
$$P = C^d \bmod n$$

where

$$ed \equiv 1 \text{ (mod some complicated number)}$$

Unlike RSA, $n$ is not defined in terms of two large primes, it must remain part of the secret key. If someone had $e$ and $n$, they could calculate $d$. Without knowledge of $e$ or $d$, an adversary would be forced to calculate

$$e = \log_P C \bmod n$$

We have already seen that this is a hard problem.

***Patents***

The Pohlig-Hellman algorithm is patented in the United States [722] and also in Canada. PKP licenses the patent, along with other public-key cryptography patents (see Section 25.5).